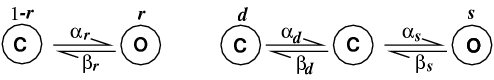
An important electrophysiological feature of subthalamic neurons is the post hyperpolarising response. When a neuron is hyperpolarised (e.g. by current injection or inhibitory synaptic input), at the end of the hyperpolarisation, a burst of activity is observed in subthalamic projection neurons (see left). This response is mediated by a low threshold calcium selective ion channel, called the **T-type** calcium channel.

The current **IT** produced from the T-type calcium channels was characterised within the Hodgkin-Huxley framework by Wang et al. (1991):

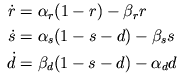
T channel equations

where **gT(max)** is the maximum T-type calcium conductance, **r** is the activation state variable, **s** is the inactivation state variable, **ECa** is the reversal potential for calcium and **V** is the neuron membrane potential. The state variables **r** and **s** are given by the following kinetic systems:



Here "C" refers to a closed state and "O" refers to an open state. The α and β variables are the forward and backward rate constants from one state to another; they are voltage dependent functions specified in Wang et al.(1991). Note that inactivation is a three state kinetic process, with fast (**s**) and slow (**d**) components).

The kinetic schemes translate to three differential equations:



We would like to add this channel to our model. Sadly this cannot be done with the programming language *hoc* that we have used in the previous tutorial parts. Instead we have to learn the **Model Description Language**(*NMODL*) provided for defining additional distributed membrane mechanisms such as ion channels and calcium pumps and point processes such as synapses.

**NMODL and NEURON**

A membrane mechanism description using *NMODL* is laid out in a text file. The *NEURON* interpreter cannot read this file directly as it can with *hoc* files. Instead, the *NMODL* file has to be *compiled* into a form that *NEURON* can use. Suppose that we have created a file *CaT.mod* containing our description of the T-type calcium channel in *NMODL*. How you incorporate this new mechanism into *NEURON* depends on what operating system you are using.

**NMODL**

**The NEURON block**

The NEURON block is the public interface of the mechanism. It tells the *hoc* interpreter how to refer to the mechanism and what variables it can see or change. The structure of the block is as follows:

NEURON {

SUFFIX *suffix*

USEION *ions...* READ *vars...* WRITE *vars...*

RANGE *var,var,...*

GLOBAL *var,var,...*

}

The first step is to identify this mechanism from all other membrane mechanisms when referencing it from the *hoc* language. This is done through the SUFFIX statement of the block. Access to all variables in this mechanism from the *hoc* file is then done using the suffix. For example, we will call this channel mechanism "CaT", so to access variables in the mechanism from *hoc* we use *var*\_CaT (where *var* is a variable in this mechanism).

The USEION statement specifies what ions this channel mechanism uses. There are three ions *NEURON* knows about, na, k, ca, however, others may also be defined via this statement. *NEURON* *can* keep track of the intracellular and extracellular concentrations of each ion. (Typically, these are actually concentrations in small shells around either side of the membrane.) Dealing with ions is difficult, because more than one mechanism may affect a particular ion. For example, we may have more than one calcium channel mechanism. Therefore, when dealing with ions use **exactly** the same name used in all other mechanisms.

The READ modifier lists ionic variables needed in calculating the ionic channel current (usually the equilibrium potential, or concentration). The WRITE modifier lists what ionic variables are calculated in this mechanism (usually the current). In our example we use:

USEION ca READ eca WRITE ica

eca is the equilibrium potential for ion ca, and ica is the calcium current, to be calculated in this mechanism.

You probably expect that since we have just introduced ica, a calcium current, *NEURON* will automatically adjust the intra- and extracellular calcium concentrations. **It doesn't** (though it can; [the box](https://web.mit.edu/neuron_v7.4/nrntuthtml/tutorial/tutD.html#ions) contains more details.) One way to think about this is that in this tutorial, we have deliberately chosen not to model calcium accumulation adjacent to the membrane, either intracellularly or extracellularly.

**Note on how *NEURON* deals with ions**

*NEURON* does not change the calcium *concentrations* automatically. To do this, we would need another mechanism defined in an *NMODL* file that would WRITE cai and/or cao, the intra- and extracellular calcium concentrations. However this mechanism would need to know the total calcium current ica originating from our CaT mechanism and any other mechanisms affecting calcium current. *NEURON* provides a means of doing this, but it is outside the scope of this tutorial.

The RANGE statement makes the following variables visible to the *NEURON* interpreter and specifies that they are be functions of position. For example, the maximum channel conductance should be a *RANGE* variable, since it can be different at different points on a neuron.

The GLOBAL statement specifies variables that are always the same for the mechanism. This mechanism does not have any GLOBAL variables. Our final NEURON block now has the form:

NEURON {

SUFFIX CaT

USEION ca READ eca WRITE ica

RANGE gmax

}

We have called the maximum channel conductance variable gmax.

**The PARAMETER block**

The PARAMETER block specifies variables that:

* are not changed as a result of the calculations in the mechanism;
* are (generally) constant throughout time; and
* can be changed by the user from the *hoc* interface or the GUI.

We can see from the equation: T channel equations

hat in calculating this channel current our model will use the voltage v, the maximum conductance we declared in the NEURON block above, gmax and the calcium equilibrium potential, eca. Of these only gmax satisfies the conditions above; v is calculated at every time step by *NEURON* rather than being specified by the user, and eca may also be calculated at each time step depending on the calcium concentrations. The PARAMETER block is:

PARAMETER {

gmax = 0.002 (mho/cm2)

}

For each parameter, we specify the name of the parameter, its default value and its units (in parentheses).

We change the value of gmax in a segment in which CaT is inserted from *hoc* as follows:

soma gmax\_CaT = 0.001

**The ASSIGNED block**

The ASSIGNED block declares variables that are either:

* calculated by the mechanism itself or
* computed by *NEURON*.

Variables that this mechanism will compute are the calcium current ica, and variables for the rate equations ralpha, rbeta, salpha etc. The variables that the mechanism uses that are computed by *NEURON* are the membrane potential v and the calcium equilibrium potential eca. The ASSIGNED block is:

ASSIGNED {

v (mV)

eca (mV)

ica (mA/cm2)

ralpha (/ms)

rbeta (/ms)

salpha (/ms)

sbeta (/ms)

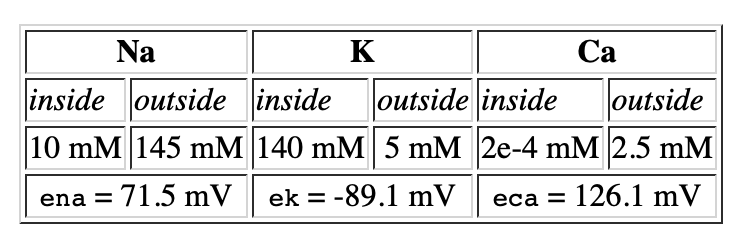
dalpha (/ms)

dbeta (/ms)

}

**Note on equilibrium potentials**

As mentioned above eca is calculated by *NEURON*. How and when it is calculated depends on the particular mechanisms dealing with ions that are inserted in a section. In this tutorial, although there is a calcium current ica, we have deliberately chosen not to represent calcium accumulation adjacent to the membrane, either intracellularly or extracellularly (see the Box: [Note on how NEURON deals with ions](https://web.mit.edu/neuron_v7.4/nrntuthtml/tutorial/tutD.html#ions)). So in our case *NEURON* use a built-in, i.e. "default", eca. But *NEURON*'s default values for eca, ena, and ek are not appropriate for our mammalian subthalamic nucleus. We want to use typical mammalian values like the following (from Johnston & Wu, 1999):

****

where the equilibrium potentials ena, ek, and eca are calculated at 37 degrees Celsius according to the Nernst equation. We will set all of these equilibrium potentials as our model has channels using each of these ion species. These should be set **after** the channel mechanisms are inserted into a section, and must be set for each section that has channels using that ionic species inserted. In our model, only the soma has such channels. We can modify our SThcell template, adding these equilibrium potential parameter values to the soma block:

soma {

nseg = 1

diam = 18.8

L = 18.8

Ra = 123.0

insert hh

ena = 71.5

ek = -89.1

gnabar\_hh=0.25

gl\_hh = .0001666

el\_hh = -60.0

insert CaT

eca = 126.1

}

Remember, if you insert channels into other sections, you must modify the appropriate equilibrium potentials in that section to make them consistent with the soma section.

**The STATE block**

The STATE block declares state variables. One type of state variable are the variables that we are trying to solve for in kinetic schemes. There are three state variables in our kinetic channel model, r, s and d. The STATE block is:

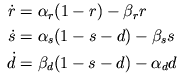
STATE {

r s d

}

**The heart of the mechanism**

We now come to the heart of the mechanism. We wish to calculate the values of the variables r, s and d in order to calculate the calcium current from the above equation. These are given by the three kinetic differential equations:



We must first calculate each of the functions ralpha, rbeta, salpha, sbeta, dalpha and dbeta. These voltage dependent functions are specified by Wang et al.. We can create a PROCEDURE to calculate these functions. This allows us to calculate the equations at any time by simply using our defined procedure. A procedure is defined using the following format:

PROCEDURE *name*(*vars*) {

*calculations...*

}

In our case, we need a procedure that takes the current voltage v and calculates values of the variables ralpha, rbeta etc. We will call our procedure settables. [Note the actual functions are taken from Wang et al. (1991)]

PROCEDURE settables(v (mV)) {

LOCAL bd

ralpha = 1.0/(1.7+exp(-(v+28.2)/13.5))

rbeta = exp(-(v+63.0)/7.8)/(exp(-(v+28.8)/13.1)+1.7)

salpha = exp(-(v+160.3)/17.8)

sbeta = (sqrt(0.25+exp((v+83.5)/6.3))-0.5) \*

(exp(-(v+160.3)/17.8))

bd = sqrt(0.25+exp((v+83.5)/6.3))

dalpha = (1.0+exp((v+37.4)/30.0))/(240.0\*(0.5+bd))

dbeta = (bd-0.5)\*dalpha

}

As these functions will need to be reevaluated at each time step (as the voltage is changing), it is more computationally efficient to create a table of values calculated at closely spaced voltages at the start of a simulation, and use table lookup with linear interpolation based on the current voltage (memory is cheaper than computation). This can be done by simply adding a TABLE line to the procedure. The TABLE command has the form:

TABLE *funcs* DEPEND *vars* FROM *lowest* TO *highest* WITH *steps*

Where *funcs*, are the variables representing the functions to create tables for (e.g. the alpha and beta function variables) and *vars* are the variables, that if they change value then all tables must be recalculated. In our case, *lowest* and *highest* are the lowest and highest values of the voltage we make the tables over, with *steps* steps between them. Our procedure now looks like:

PROCEDURE settables(v (mV)) {

LOCAL bd

TABLE ralpha, rbeta, salpha, sbeta, dalpha, dbeta

FROM -100 TO 100 WITH 200

ralpha = 1.0/(1.7+exp(-(v+28.2)/13.5))

rbeta = exp(-(v+63.0)/7.8)/(exp(-(v+28.8)/13.1)+1.7)

salpha = exp(-(v+160.3)/17.8)

sbeta = (sqrt(0.25+exp((v+83.5)/6.3))-0.5) \*

(exp(-(v+160.3)/17.8))

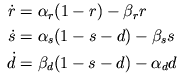
bd = sqrt(0.25+exp((v+83.5)/6.3))

dalpha = (1.0+exp((v+37.4)/30.0))/(240.0\*(0.5+bd))

dbeta = (bd-0.5)\*dalpha

}

Now, to calculate all the alpha and beta functions we simply call the procedure settables. The alpha and beta functions are used by our differential equations:



These equations can be directly specified in a DERIVATIVE block. The DERIVATIVE block requires a name, and we will call ours states.

DERIVATIVE states {

settables(v)

r' = ((ralpha\*(1-r)) - (rbeta\*r))

d' = ((dbeta\*(1-s-d)) - (dalpha\*d))

s' = ((salpha\*(1-s-d)) - (sbeta\*s))

}

Each time *NEURON* calculates the differential equations, the alpha and beta variables must be updated, so the first line calls our procedure settables with the current voltage v. Then each differential equation is specified.

Our BREAKPOINT is the top level mechanism calculation block that simply solves the differential equations and calculates the calcium current.

BREAKPOINT {

SOLVE states METHOD cnexp

ica = gmax\*r\*r\*r\*s\*(v-eca)

}

The current is calculated directly from the equation:

T channel equations

The SOLVE statement refers to the states defined in our DERIVATIVE block. The METHOD cnexp part of the line tells *NEURON* to use the "cnexp" method of integration. This method is suitable for mechanisms of the form

y' = f(V,y)

where *f* is linear in *y* and involves no other states. Hodgkin-Huxley-type equations fall into this category.

The final block we must specify is the INITIAL block. This block is used to set the state variables r, d and s to to their initial values. Usually these are the values of the state variables at equilibrium. The INITIAL block first calls the procedure settables with the present voltage to calculate the values of the alpha and beta functions. These in turn are then used to calculate the initial state variables. The INITIAL block is:

INITIAL {

settables(v)

r = ralpha/(ralpha+rbeta)

s = (salpha\*(dbeta+dalpha) - (salpha\*dbeta))/

((salpha+sbeta)\*(dalpha+dbeta) - (salpha\*dbeta))

d = (dbeta\*(salpha+sbeta) - (salpha\*dbeta))/

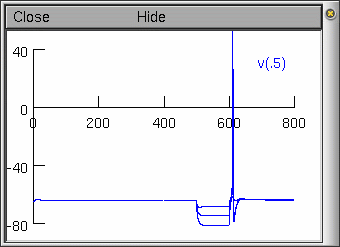
((salpha+sbeta)\*(dalpha+dbeta) - (salpha\*dbeta))

}

Note the values for r, d and s are derived from the differential equations at equilibrium.

One final note, you may notice the commands UNITSON and UNITSOFF in the file. These activate the units checking (e.g. mV, mA etc.) and deactivate it respectively.

Now if we inject a hyperpolarising current in one of our neurons, we now observe a post-hyperpolarising T-type response:



With current injections at -0.1, -0.2, and -0.3 nA.

**Note:** The action potential is unrealistic for mammalian subthalamic cells (it is still based on our HH squid axon channels). We leave it to the reader to go on and develop channels to underlie an action potential more characteristic of the mammalian subthalamic projection neuron.

**Understanding MOD Files for Calcium**

MOD files define **how ion channels, pumps, and buffers work in NEURON simulations**.

Each MOD file follows a specific structure:

**Key Sections of MOD Files**

1. **NEURON Block: Defines what the mechanism does**

**NEURON {**

**SUFFIX cdp20N\_FD2 // Name of the mechanism**

**USEION ca READ cao, cai, ica WRITE cai // Reads & modifies calcium (ca) ion concentration**

**RANGE ica\_pmp**

**GLOBAL vrat, TotalPump**

**}**

• USEION ca → This mechanism **reads and writes calcium ion concentration** (cai is internal, cao is external).

• RANGE ica\_pmp → This allows **calcium pump current to be accessed in simulations**.

• GLOBAL TotalPump → **Defines pump strength**.

2. **PARAMETER Block: Defines adjustable values**

**PARAMETER {**

**kpmp1 = 3e3 (/mM-ms) // Calcium pump binding rate**

**kpmp2 = 1.75e1 (/ms) // Pump unbinding rate**

**kpmp3 = 7.255e1 (/ms) // Rate of calcium extrusion**

**TotalPump = 1e-15 // Defines pump density**

**vmax = 0.1**

**Kp = 2.7e-3 (mM) // Calcium affinity of the pump**

**}**

• kpmp1 → **How fast the pump binds calcium**.

• kpmp2 → **How fast calcium unbinds from the pump**.

• kpmp3 → **How quickly calcium is pumped out**.

• TotalPump → **Controls total pump density** (setting to 0 disables the pump).

• vmax → **Maximum rate at which calcium is removed**.

**ASSIGNED Block: Defines calculated variables**

**ASSIGNED {**

**ica (mA/cm2) // Calcium current**

**ica\_pmp (mA/cm2) // Calcium current due to pumps**

**parea (um) // Pump surface area**

**}**

• These **store intermediate values** used in the equations.

4. **BREAKPOINT Block: Main computation**

**BREAKPOINT {**

**SOLVE state METHOD sparse**

**}**

• This is **where the calcium pump equations are solved**.

• Uses a STATE function to update calcium concentrations.

5. **DERIVATIVE Block: Defines how calcium changes over time**

**DERIVATIVE state {**

**cai' = - (ica\_pmp / (2 \* FARADAY \* parea)) // Calcium removal equation**

**}**

• cai' represents **the rate of change of intracellular calcium concentration**.

• ica\_pmp is **calcium pump activity**.